Continuations, proofs and tests[*]

Stefano Guerrini[†] Andrea Masini[‡]

February 1, 2008

Abstract

Continuation Passing Style (CPS) is one of the most important issues in the field of functional programming languages, and the quest for a primitive notion of types for continuation is still open.
Starting from the notion of "test" proposed by Girard, we develop a notion of test for intuitionistic logic.
We give a complete deductive system for tests and we show that it is good to deal with "continuations". In particular, in the proposed system it is possible to work with Call by Value and Call by Name translations in a uniform way.

1 Introduction

Since the seminal works of Fischer [5] and Plotkin [14] continuations have become central in the study and implementation of functional languages. In particular, by defining the Call-by-Name and Call-by-Value translations of λ-calculus, Plotkin posed the basis of CPS (Continuation Passing Style) transforms.

After these initial milestones, any overview of CPS, even if very short, cannot avoid to mention the fundamental work that Felleisen, Friedman, Kohlbecker, Duba, and Sabry carried out for type free functional languages. Felleisen et al. [4] were the first to axiomatize the so called control operators—`call/cc` and other similar operators of lisp-like languages. Some years later, Sabry and Felleisen [15] were the first to prove completeness results for CPS transforms of type-free functional languages.

While Felleisen and his collaborators were developing the theory of CPS transforms, several researchers began to investigate if it was possible to explain CPS by means of some known logic, in the style of the well-known correspondence between intuitionistic logic and types and computations of functional languages.

[*]Partially supported by the MIUR PRIN "Logical foundations of abstract programming languages".

[†] Dipartimento di Informatica, Università Roma La Sapienza - Via Salaria, 113 - 00198 Roma - Italy - email: `guerrini@dsi.uniroma1.it`

[‡] Dipartimento di Informatica, Università di Verona - Ca' Vignal 2, strada le Grazie, 15 - 37134 Verona - Italy - email: `andrea.masini@univr.it`

Griffin [8] was the first one to give a partial answer to this question by proposing classical logic as a type system for a simplified version of Scheme. The main idea of Griffin was to use Reductio ad Absurdum to explain Felleisen's control operator $\mathcal{C}$.

Although the work of Griffin opened new perspectives in the use of classical logic for the study of programming languages, it left unanswered several questions. First of all, "classical logic seemed not to have a clear computational interpretation" because of the lack of confluence of its "standard" natural deduction formulation or, as observed by Joyal in categorical terms, because of the collapse of proofs in the standard categorical semantics of classical logic. But, what does it happen if we change the rules of the game, namely the "formulation of the logic"?

In [12, 13], Parigot showed that a drastically different formulation of classical logic, the so-called $\lambda\mu$-calculus, allows to give a computational meaning to the cut elimination procedure—$\lambda\mu$-calculus enjoys the nice computational properties of λ-calculus: strong normalization and confluence.

After the introduction of the $\lambda\mu$-calculus, several researchers tried to show that it might have been a foundational calculus for CPS (e.g., de Groote [3]). Unfortunately, such a research did not led to the expected results: it pointed out many analogies between $\lambda\mu$-calculus and continuations, but, at the same time, it showed that $\lambda\mu$-calculus fails to give a precise definition of basic control operators. Indeed, it showed that even if the μ-reduction has a "continuation flavor", it is not the right reduction for CPS transformed programs. In spite of these negative results, $\lambda\mu$-calculus remains one of the most important logical calculi for CPS.

In [9], Hofmann and Streicher proposed a categorical continuation model for a Call-by-Name version of $\lambda\mu$-calculus. As already done by Griffin, Hofmann and Streicher used classical logic, and in particular the Reduction ad Absurdum principle, to define the meaning of CPS. Anyhow, while Griffin used Reduction ad Absurdum to give a type assignments to CPS terms, Hofmann and Streicher embedded Reduction ad Absurdum in the construction of the semantic domains for the interpretation of CPS.

Subsequently, in [17], Streicher and Reus extended the ideas in [9] giving a categorical semantics of a Call-by-Value λ-calculus equipped with the control operator $\mathcal{C}$ of Felleisen.

Recently, a very interesting analysis of CPS in terms of proof/type theoretical methods has been proposed by Curien and Herbelin: in [1], they have shown that the most known CPS translations may be obtained by means of a suitable translation between $\lambda\mu$-calculus and a new formulation of λ-calculus plus control operators. Another interesting proof-theoretical contribution is the work of Ogata [10], who related a Call-by-Value normalization of the $\lambda\mu$-calculus with the cut-elimination of one of the logical systems proposed by Danos, Joinet and Schellinx for the analysis/embedding of classical logic trough/into linear logic.

The results of Hoffman, Streicher and Reus [9, 17] were the natural background for the introduction of Selinger's Control Categories [16], perhaps, one the most important steps towards a semantic/logical explanation of CPS. In fact, Control Categories were the first model of $\lambda\mu$-calculus in which Call-by-Value and Call-by-name have a uniform interpretation.

Few years after the work of Selinger, Führmann and Thielecke [6] presented a quite different approach to the semantics of CPS—even if, restricted to the case of an idealized Call-by-Value functional language. In particular, they proposed both a type theoretical and a categorical semantics approach to CPS, and studied in detail the CPS transforms.

1.1 Interaction

The problem of a satisfactory logical explanation of continuations remains open.

Quite naturally, one may observe that any solution to such a problem must base on a deep interaction between programs and computations. Therefore, a good question is: in logic, is there any explicit notion of "interaction" that could be used in CPS? The positive answer, in our opinion, is in the proposal of Girard for an "interactive approach to logic".

The key point of Girard [7] is the idea that the meaning of proofs does not reside in some external world called the "semantics of the proofs"; the meaning of proofs is described by the interaction between proofs and some dual objects that Girard name *tests*.

The proof/test duality introduced by Girard can be understood in terms of a game between a player and an opponent. A *proof* is a sequence of arguments used by the player to assert that, moving from a given set of premises, the ending formula (or sequent) holds. Then, what is the dual of a proof? A *test* is a sequence of arguments used by the opponent to confute the provability of a formula (or sequent).

What does it happen if the player asserts that a formula is provable while the opponent says that such a formula is not provable? If the system is not trivial—and by the way, we are interested in such a case only—someone is cheating and we need a way to validate the arguments used in a proof/test. In this kind of game there is no referee and we cannot resort to any external argument. So, the only way that we have to discover who is cheating is by counterposing the proof proposed by the player to the test proposed by the opponent. The interaction between the two derivations (cut-elimination) will lead to discover where the arguments of the player or of the opponent fail.

Another important issue is constructiveness: if we do not want to exit outside our computational world, both proof and tests must be constructive.

The BHK interpretation asserts that:

> A *proof* π of $A \to B$ is a (constructive) transformation from a proof of A to a proof of B.

In particular, if there is no proof of A, the transformation π is the empty map, and we do not have any argument to refute it. On the other hand, given a proof of A, the transformation π leads to a proof of B that we can attempt to refute. Therefore, it is quite natural to assert that:

a *test* t of $\mathsf{A} \rightarrow \mathsf{B}$ is a pair (π, τ) such that:

(i) π is a proof of A and

(ii) τ is a test of B.

Asking at the same time that:

> a proof is a "failure" of a test and a test is a "failure" of a proof.

Such a notion of duality has been our starting point in the development of a type system for continuations.

1.2 Our proposal

We propose a new calculus, the ptq-calculus, characterized by the relevant properties summarized below.

1. The ptq-calculus bases on a *general primitive notion of continuation/test.* That *unique notion of continuation* is suitable to deal with both Call-by-Value and Call-by-Name languages.
2. The ptq-calculus is equipped with a *deterministic* one step lazy reduction relation: *the calculus is, per se, neither Call-by-Value nor Call-by-Name.* A term is either in normal form or a redex.
3. Even if the ptq-calculus is neither Call-by-Value nor Call-by-Name, *it can code (in a sound and complete way) Call-by-Value and Call-by-Name* λ-*calculi.*

2 Proof theoretical motivations

The technical details of the ptq-calculus will be presented in section 3. In this section, we shall give a detailed and informal explanations of the proof theoretical motivations that have led us to the calculus of continuations.

2.1 The starting point: classical logic

In the introduction, we have already seen that

- a test t of $\mathsf{A} \rightarrow \mathsf{B}$ is a pair (π, τ), where
  1. π is a proof of A

     and
  2. τ is a test of B;
- we have the following proof/test duality:
  - a proof is a **failure** of a test;

– a test is a **failure** of a proof.

Starting from these basic properties, the definition of test can be extended in order to obtain a sound and complete proof system. The system has two kind of formulas:

- *proof formulas*, denoted by A^p;
- *test formulas*, denoted by A^t.

The judgments of the calculus are sequents of the form $\Gamma \vdash \alpha$, where Γ is a set of proof and test formulas, and α is either empty, or a proof formula, or a test formula.

The following are the rules of the proof system—let us call it KT.

$$\Gamma, \mathsf{A}^\mathsf{p} \vdash \mathsf{A}^\mathsf{p} \qquad \Gamma, \mathsf{A}^\mathsf{t} \vdash \mathsf{A}^\mathsf{t}$$

$$\frac{\Gamma, \mathsf{A}^\mathsf{p}, \mathsf{B}^\mathsf{t} \vdash}{\Gamma \vdash \mathsf{A} \to \mathsf{B}^\mathsf{p}} \qquad \frac{\Gamma \vdash \mathsf{A}^\mathsf{p} \quad \Gamma \vdash \mathsf{B}^\mathsf{t}}{\Gamma \vdash \mathsf{A} \to \mathsf{B}^\mathsf{t}}$$

$$\frac{\Gamma, \mathsf{A}^\mathsf{t} \vdash}{\Gamma \vdash \mathsf{A}^\mathsf{p}} \qquad \frac{\Gamma, \mathsf{A}^\mathsf{p} \vdash}{\Gamma \vdash \mathsf{A}^\mathsf{t}}$$

$$\frac{\Gamma \vdash \mathsf{A}^\mathsf{t} \quad \Gamma \vdash \mathsf{A}^\mathsf{p}}{\Gamma \vdash}$$

It is quite easy to prove that KT is a presentation of classical logic.

Proposition 1. *The sequent $\Gamma \vdash \Delta$ is derivable in LK iff the judgment $\Gamma^\mathsf{p}, \Delta^\mathsf{t} \vdash$ is derivable in KT.*

As a direct consequence, A *is classically valid iff there exists a derivation of* $\vdash \mathsf{A}^\mathsf{p}$ *in* KT.

2.2 Leaving the classical world ...

It is immediate to observe that the proof-test duality is reminiscent of the well-known de Morgan duality: if we translate each A^p as A and each A^t as $\neg\mathsf{A}$ in the above proposed deductive system, we obtain a set of admissible rules for LK. But, in spite of such a connection, the proof-test duality does not introduce any kind of classical principle, and in fact it will be used in an *intuitionistic* setting.

Looking carefully at the proposed system, it is possible to observe that:

- a premise A^t morally corresponds to a conclusion A;
- a conclusion A^t morally corresponds to a premise A;
- a premise A^p directly corresponds to a premise A;
- a conclusion A^p directly corresponds to a conclusion A.

As a matter of fact, it is possible to translate each judgment G of KT in an ordinary sequent $\mathsf{S} = (\mathsf{G})^+$ of LK:

- $(\Gamma^p, \Delta^t \vdash)^+ = \Gamma \vdash \Delta$;
- $(\Gamma^p, \Delta^t \vdash A^p)^+ = \Gamma \vdash \Delta, A$;
- $(\Gamma^p, \Delta^t \vdash A^t)^+ = \Gamma, A \vdash \Delta$;

Such a translation, when applied to the rules of KT, produces the rules:

$$\Gamma, A \vdash \Delta, A \qquad \frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B} \qquad \frac{\Gamma \vdash \Delta, A \quad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta} \qquad \frac{\Gamma, A \vdash \Delta \quad \Gamma \vdash \Delta, A}{\Gamma \vdash \Delta}$$

which are the standard LK rules of classical logic.

Now, let us consider *minimal logic*, i.e., the system of types for simply typed λ-calculus. We know that minimal logic is obtained by means of a structural constraint: *the sequents must have exactly one conclusion.*

If we want that the $(\ )^+$ translation produces minimal logic sequents, we must constrain the structure of derivable judgments in such a way that:

C1. for each judgment $\Gamma \vdash A^p$, the set Γ does not contain test formulas;

C2. for each judgment $\Gamma \vdash$, the set Γ contains exactly one test formula;

C3. for each judgment $\Gamma \vdash A^t$, the set Γ contains exactly one test formula.

The simpler way to obtain a deductive system such that all the derivable judgments obey to the constraints (C1), (C2) and (C3) is to impose a *linear discipline for test formulas*, as in the deductive system below—let us call it mT.

$$\Gamma^p, A^p \vdash A^p \qquad \Gamma^p, A^t \vdash A^t$$

$$\frac{\Gamma^p, A^p, B^t \vdash}{\Gamma^p \vdash A \to B^p} \qquad \frac{\Gamma^p \vdash A^p \quad \Gamma^p, C^t \vdash B^t}{\Gamma^p, C^t \vdash A \to B^t}$$

$$\frac{\Gamma^p, A^t \vdash}{\Gamma^p \vdash A^p} \qquad \frac{\Gamma^p, A^p, B^t \vdash}{\Gamma^p, B^t \vdash A^t}$$

$$\frac{\Gamma^p, B^t \vdash A^t \quad \Gamma^p \vdash A^p}{\Gamma^p, B^t \vdash}$$

It is possible to prove that mT is a presentation of minimal logic.

Proposition 2. *The sequent* $\Gamma \vdash A$ *is derivable in minimal logic iff the judgment* $\Gamma^p \vdash A^p$ *is derivable in* mT.

2.2.1 ... and approaching to continuations.

In the perspective of the development of a type theory for continuations, we think that the notion test described above is the right one. Therefore, let us propose an extension of the standard Curry-Howard isomorphism, by providing a correspondence between:

- *deductions of* A^p *and programs of type* A^p;
- *deductions of* A^t *and continuations of type* A^t.

As a first step, let us transform mT into a type system.

$$\Gamma^p, x : A^p \vdash x : A^p \qquad\qquad \Gamma^p, k : A^t \vdash k : A^t$$

$$\frac{\Gamma^p, x : A^p, k : B^t \vdash u}{\Gamma^p, \vdash \lambda\langle x^{A^p}, k^{B^t}\rangle.u : A \to B^p} \to^p \qquad \frac{\Gamma^p \vdash p : A^p \quad \Gamma^p, k : C^t \vdash t : B^t}{\Gamma^p, k : C^t \vdash \langle p, t\rangle : A \to B^t} \to^t$$

$$\frac{\Gamma^p, k : A^t \vdash u}{\Gamma^p \vdash \lambda k^{A^t}.u : A^p} \lambda^p \qquad \frac{\Gamma^p, x : A^p, k : B^t \vdash u}{\Gamma^p, k : B^t \vdash \lambda x^{A^p}.u : A^t} \lambda^t$$

$$\frac{\Gamma^p, k : B^t \vdash t : A^t \quad \Gamma^p \vdash p : A^p}{\Gamma^p, k : B^t \vdash t \bullet p} @$$

The reduction rules for terms that naturally arise from the above syntax are:

$$\begin{array}{rcl} \langle p, t\rangle \bullet \lambda\langle x, k\rangle.u & \to & u[p/x, t/k] \\ \lambda x.u \bullet p & \to & u[p/x] \\ t \bullet \lambda k.u & \to & u[t/k] \end{array}$$

Unfortunately, such a system has the main defect of any naive term system associated to classical logic: it is *non-confluent*. In fact,

$$\begin{tikzcd} & \lambda x.u' \bullet \lambda k.u'' \arrow[dl] \arrow[dr] & \\ u'[\lambda k.u''/x] & & u''[\lambda x.u'/k] \end{tikzcd}$$

and there is no general way to close the diagram.

The non-confluence of the calculus cannot be solved by imposing a fixed reduction strategy for $\lambda x.u' \bullet \lambda k.u$. There is not a standard way to make a choice between $\lambda x.u' \bullet \lambda k.u \to u'[\lambda k.u''/x]$ and $\lambda x.u' \bullet \lambda k.u \to u''[\lambda x.u'/k]$. Each of the two possible choices implies serious problem in normalization. It is exactly the problem of reducing a cut between $\neg A$ and A in classical logic.

Moreover, there is a "programming language" reason forcing to reject the choice of fixed reduction strategy for $\lambda x.u' \bullet \lambda k.u$.

By assuming that $\lambda x.u' \bullet \lambda k.u$ always reduces to $u'[\lambda k.u''/x]$, we impose that *continuation are "functions"*; on the other hand, by assuming $\lambda x.u' \bullet \lambda k.u$ always reduce to $u''[\lambda x.u'/k]$, we impose that *continuations are "arguments"*. But, unfortunately, both that choices do not agree with the continuation passing style translations of Call-by-Value and Call-by-Name functional languages (e.g., see [14]). In other words, *"we cannot statically determine whether a continuation is an argument or a function"*.

In order to solve the problem of composition between tests (continuations) and proofs (programs) we propose:

1. a "new class of types" A^q, where is any intuitionistic type, s.t. A^q is a subtype of A^p;
2. two different ways for composing a program p and a continuation t:
   - a standard composition pt, in which the continuation is an argument;
   - a dynamic composition $t;p$, where which term plays the role of the argument is not statically fixed (it could be either t or p), depending on the shape of t and p.

3 The ptq-calculus

The set of the *type expressions* is given by the following grammar:

X	$::=$	$X_1 \mid \dots \mid X_k$	base types
A	$::=$	$X \mid A \rightarrow A$	intuitionistic types
P	$::=$	A^p	proof types
T	$::=$	A^t	test types
Q	$::=$	A^q	q-proof types
W	$::=$	$P \mid T \mid Q$	types

The set of the ptq-*term expressions*, or ptq-*terms* for short, is defined by the following grammar:

x	$::=$	$x_0, x_1, \dots$	p-variables
k	$::=$	$k_0, k_1, \dots$	t-variables
p	$::=$	$x \mid \lambda\langle x, k\rangle.u \mid \lambda k.u$	p-terms
t	$::=$	$* \mid k \mid \langle p, t\rangle \mid \lambda x.u$	t-terms
q	$::=$	$\bar{\lambda} k.u$	q-terms
u	$::=$	$t;p \mid qt$	e-terms

In order to simplify the treatment of substitution, we shall assume to work modulo variable renaming, i.e., term-expressions are equivalence classes modulo α-conversion. Substitution up to α-equivalence is defined in the usual way.

3.1 The type system

A *type environment* Ξ is either a set Γ or a pair of sets $\Gamma \rhd \delta$, where Γ is a (possibly empty) set $x_1 : P_1, \ldots, x_n : P_n$ of typed p-variables, such that all the variables x_i are distinct, and δ is either a singleton $k : A^t$ (a typed t-variable) or a singleton $* : A^t$ (a typed constant).

A *judgment* is an expression $\Xi \vdash \xi$, where Ξ is a type environment, ξ is a typed ptq-term expression, and all the free variables in ξ occur in Γ.

The set of the *well-typed* ptq*-terms* and the set of the *well-typed judgments* are defined by the type system in Figure 1, where A and B are metavariables ranging over intuitionistic types. (In the following, when not otherwise specified, the metavariables $A, B, C, \ldots$ will range over intuitionistic types.)

$$\Gamma, x : A^p \vdash x : A^p \qquad \Gamma \rhd k : A^t \vdash k : A^t$$

$$\Gamma \rhd * : A^t \vdash * : A^t$$

$$\frac{\Gamma, x : A^p \rhd k : B^t \vdash u}{\Gamma \vdash \lambda\langle x, k\rangle.u : A \to B^p} \to^p \qquad \frac{\Gamma \vdash p : A^p \quad \Gamma \rhd \delta \vdash t : B^t}{\Gamma \rhd \delta \vdash \langle p, t\rangle : A \to B^t} \to^t$$

$$\frac{\Gamma \rhd k : A^t \vdash u}{\Gamma \vdash \lambda k.u : A^p} \lambda^p \qquad \frac{\Gamma, x : A^p \rhd \delta \vdash u}{\Gamma \rhd \delta \vdash \lambda x.u : A^t} \lambda^t$$

$$\frac{\Gamma \rhd k : A^t \vdash u}{\Gamma \vdash \overline{\lambda} k.u : A^q} \overline{\lambda}^q$$

$$\frac{\Gamma \vdash p : A^p \quad \Gamma \rhd \delta \vdash t : A^t}{\Gamma \rhd \delta \vdash t; p} @^p$$

$$\frac{\Gamma \vdash q : A^q \quad \Gamma \rhd \delta \vdash t : A^t}{\Gamma \rhd \delta \vdash qt} @^q$$

Figure 1: ptq-type system

By inspection of the type system in Figure 1, we see that:

- in well-typed ptq-terms, t-variables are linear;
- in a well-typed p-term/q-term there are no free occurrences of t-variables and no occurrences of the constant $*$;
- in a well-typed t-term/e-term there is one free occurrence of a t-variable or, alternatively, one occurrence of the constant $*$, that in any case cannot be enclosed by a t-variable binder.

Summing up, in order to construct well-typed terms, we suffice one name for t-variables. Therefore, in the following, we shall assume that all the occurrences of t-variables have name k.

Fact 3. *The* ptq*-type system has the substitution property.*

1. *For any well-typed* $\Gamma \;\rhd\; \xi \vdash t' : A^t$*, we have that*
   (a) *for every well-typed* $\Gamma, \Delta \;\rhd\; k : A^t \vdash t : B^t$ *or* $\Gamma, \Delta \;\rhd\; * : A^t \vdash t : B^t$*, the corresponding* $\Gamma, \Delta \;\rhd\; \xi \vdash t[t'/k] : B^t$ *or* $\Gamma, \Delta \;\rhd\; \xi \vdash t[t'/*] : B^t$ *is well-typed;*
   (b) *for every well-typed* $\Gamma, \Delta \;\rhd\; k : A^t \vdash u$ *or* $\Gamma, \Delta \;\rhd\; * : A^t \vdash u$*, the corresponding* $\Gamma, \Delta \;\rhd\; \xi \vdash u[t'/k]$ *or* $\Gamma, \Delta \;\rhd\; \xi \vdash u[t'/*] : B^t$ *is well-typed.*
2. *For any well-typed* $\Gamma \vdash p' : A^p$*, we have that*
   (a) *for every well-typed* $\Gamma, \Delta, x : A^p \;\rhd\; \xi \vdash t : B^t$*, then* $\Gamma, \Delta \;\rhd\; \xi \vdash t[p'/x] : B^t$ *is well-typed;*
   (b) *for every well-typed* $\Gamma, \Delta, x : A^p \vdash u$ *or* $\Gamma, \Delta, x : A^p \vdash p : B^p$ *or* $\Gamma, \Delta, x : A^p \vdash q : B^q$*, the corresponding* $\Gamma, \Delta \vdash u[p'/x]$ *or* $\Gamma, \Delta \vdash p[p'/x] : B^p$ *or* $\Gamma, \Delta \vdash q[p'/x] : B^q$ *is well-typed.*

A term is t-closed when it does not contain free occurrences of t-variables. We have already seen that every well-typed p-term/q-term is t-closed and that every t-closed well-typed t-term/e-term contains either a free occurrence of the t-variable k or an occurrence of the constant $*$ outside the scope of any t-binder. Thus, every t-closed well-typed t-term t_* or e-term u_* can be obtained by replacing $*$ for the free t-variable k in a well-typed t-term t or e-term u that does not contain any occurrence of the constant $*$, namely

$$\begin{array}{rclcrcl} t_* &=& t[*/k] & \text{with} & t_*[k/*] &=& t \\ u_* &=& u[*/k] & \text{with} & u_*[k/*] &=& u \end{array}$$

respectively.

In the following, we shall only consider well-typed ptq-terms and well-typed judgments; therefore, we shall omit to specify that a term or judgment is well-typed. When not otherwise specified, we shall always denote t-closed t-terms or e-terms with a $*$ subscript; moreover, given a t-closed t-term t_* and a t-closed e-term u_*, the terms t and u are the terms such that $t_* = t[*/k]$ and $u_* = u[*/k]$.

The $*$-composition of t-closed t-terms is defined by

$$t_* \circ t'_* = t'_*[t_*/*] = t'[t_*/k]$$

It is readily seen that the $*$-composition is associative and that $*$ is its neutral element. Such a composition is extended to

$$t_* \circ u_* = u_*[t_*/*] = u[t_*/k]$$

which corresponds to $t_* \circ (t'_*; p) = (t_* \circ t'_*); p$ and $t_* \circ q t'_* = q(t_* \circ t'_*)$.

3.2 Computations

According to a standard *lazy* approach, the reduction rules that we shall define *do not reduce inside the scope of a* λ *or* $\overline{\lambda}$ *binder and inside a pair.* Since

- a p-*term* is either a variable or begins with a λ,
- a t-*term* is either a variable or begins with a λ or it is a pair,
- a q-*term* begins with a $\overline{\lambda}$,

t-terms, p-terms and q-terms are *irreducible.*

Now, let us observe that every e-term u is the composition of irreducible terms, namely $u = t; p$ or $u = qt$, where t, p, q are irreducible. In the ptq-calculus, we do not have any kind of evaluation context, nor any notion of reduction strategy: a term (a e-term) contains at most one redex and either it is in normal form or it is the redex to be reduced. This is a main difference w.r.t. standard λ-calculi, where we may choose the order of evaluation by fixing a reduction strategy, for instance, (lazy) call-by-value or call-by-name. Moreover, even when we choose a reduction strategy, the β-redex R that we have to reduce may be deeply nested into the term T, that is $T = \mathcal{C}[R]$, where $\mathcal{C}[]$ is the evaluation context determined by the reduction strategy.

The fact that in the ptq-calculus e-terms only can be redexes corresponds to the intended interpretation that proofs are programs and tests are continuations. In particular, in order to start the execution of a program we need to pass a continuation to it, that is we have to compose the p-term corresponding to the program that we want to execute with the t-term corresponding to the continuation that we want to pass to it.

In a p-composition $u = t; p$, the execution is controlled by the shape of the t-term t (the continuation), namely

1. when t is a constant or a pair, the control passes to the p-term p. In particular,
   (a) if $p = \lambda k.u$, then p corresponds to a suspended execution that is waiting for a continuation to put in the place of the parameter k. When p is applied to the t-term t, the variable k is replaced with t in the body of p and the execution resumes;
   (b) if $p = \lambda\langle x, k\rangle.u$, then p corresponds to a suspended execution that is waiting for a program to put in the place of the parameter x and a continuation to put in the place of the parameter k. Then the term $t; p$ reduces only when $t = \langle p', t'\rangle$ is a pair; in that case, the variables x and k in u are replaced by the program p' and the continuation t', respectively, and the execution resumes;
2. when t is a λ-abstraction, the continuation corresponding to $t = \lambda x.u$ can be interpreted as a suspended execution waiting for the actual value of a parameter x. Therefore, after replacing the program p for x in u, the execution resumes.

Summing up, we have the reduction rules

$$\begin{array}{rcl} *;\lambda k.u & \to & u_* \\ \langle p,t\rangle;\lambda k.u & \to & u[\langle p,t\rangle/k] \\ \langle p,t\rangle;\lambda\langle x,k\rangle.u & \to & u[p/x,t/k] \\ \lambda x.u;p & \to & u[p/x] \end{array}$$

Let us remark that, when t and p are both λ-abstraction, $t = \lambda x.u$ is a λ-abstraction, if $p = \lambda k.u'$ is a λ-abstraction too, we might try to reduce $t;p = \lambda x.u;\lambda k.u'$ by replacing t for the variable k in u'. Unfortunately, such a reduction rule would lead to the critical pair

$$u'[\lambda k.u/x] \quad \leftarrow \quad \lambda x.u';\lambda k.u \quad \to \quad u[\lambda x.u'/k]$$

In order to avoid critical pairs, we assume that, when t is a λ-abstraction, t is the function to be applied on the argument p, independently from the shape of p. In other words, we have a dynamic reduction strategy, corresponding to a sequential policy of the kind "first fit", which summarizes in the following rule:

reducing a t-application $t;p$

1. analyze first t and then p;
2. contract the application by assuming that the first term that is "usable" as a function receives the other term as an argument.

The previous reduction rule does not suffice for our purposes. In some cases, namely for the encoding of call-by-value, we also need the rule that takes the t-term t as an argument of some kind of function constructed by abstracting the only free t-variable k in the e-term u, even when $t = \lambda x.u$ is a λ-abstraction at its turn. For this reason, in the type system, we have an abstraction $\overline{\lambda} k.u : A^{\mathsf{q}}$ that, if $k : A^{\mathsf{t}}$, construct a term $\mathsf{q} : A^{\mathsf{q}}$, whose type is not a test type, and another application qt, for which we have the only reduction rule

$$\overline{\lambda} x.u;t \quad \to \quad u[t/k]$$

Let us remark that, since any q-term is a $\overline{\lambda}$-abstraction, a term qt is always a redex.

In the previous analysis we have omitted the case in which t is a variable. The reason is that, if we take $k;\lambda k.u \to u$, we do not have $(k;\lambda k.u)[\lambda x.u'/k] \to u[\lambda x.u'/k]$, but $(k;\lambda k.u)[\lambda x.u'/k] \to u'[\lambda k.u/x]$. Because of this, we prefer to assume that the term $k;p$ is always irreducible. This also explains the role of the constant $*$. Since, such a constant cannot be bound, it cannot be interpreted as a placeholder for an arbitrary t-term and there is no problem in reducing the e-term $*;\lambda k.u$ to u_*. Anyhow, we remark that $*;\lambda\langle x,k\rangle.u$ is irreducible.

In order to complete the explanation of the role of the constant $*$, let us recall that we want to interpret a p-term p as the translation of a program. Since e-terms only are reducible, in order to start the computation of p we have to composite it with some t-term, that is to pass some continuation to the program. The simplest choice is to compose p with the constant $*$. Correspondingly, the constant $*$ plays the role of the initial continuation of the system: *the continuation that the "system" passes to the compiled code in order to start the computation.*

This assumption is fully justified by the fact that, following a Continuation Passing Style, we shall compile any λ-term into t-closed p-term. Because of this, we can also

restrict the reduction rules to t-closed terms.

The complete set of the rules of the calculus are given in Figure 2.

$$\begin{array}{rcl} *; \lambda k.u & \to & u_* \\ \langle p, t_* \rangle; \lambda k.u & \to & u[\langle p, t_* \rangle / k] \\ \langle p, t_* \rangle; \lambda \langle x, k \rangle.u & \to & u[p/x, t_*/k] \\ \lambda x.u_*; p & \to & u_*[p/x] \\ (\overline{\lambda} k.u) t_* & \to & u[t_*/k] \end{array}$$

Figure 2: The reduction rules of the ptq-calculus (restricted to t-closed terms)

As usual, we shall denote by $\xrightarrow{*}$ the transitive and reflexive closure of $\to$.

One of the standard interpretation of a continuation is as the rest of computation: the continuation passed to a program specifies how the computation must continue after the completion of the program. (For a comparison of this interpretation with the interpretation that thinks at a continuation as an evaluation context, see [2].) Accordingly, we expect that the reduction of $t; p$ starts by the reduction that mimic the execution of the program corresponding to p and that, only after the completion of that program, the execution is resumed by the continuation. In practice, we expect that $t_*; p \xrightarrow{*} t_* \circ u_* = u[t/k]$, whenever $*; p \xrightarrow{*} u_*$. However, it is readily seen that this cannot hold if $t_* = \lambda x.u'_*$ and $p = \lambda k.u''_*$.

Lemma 4 gives the exact condition under which we may get the expected replacement property: either t_* is not a λ-abstraction or, when this is not the case, during the reduction of $*; p$ we never apply the rule that reduces a term with the shape $*; \lambda k.u''_*$. Let us remark that this condition hold for the translations of λ-terms that we shall give in the paper.

Lemma 4. *Let* u_* *be an* e*-term s.t.* $u_* \xrightarrow{*} u'_*$*. Given a* t*-term* t_**,*

1. *if* t_* *is not an abstraction* $\lambda x.u''_*$*, or*
2. $u_* \xrightarrow{*} u'_*$ *without reducing any redex with the shape* $*; \lambda k.u''_*$*,*

then $t_* \circ u_* \xrightarrow{*} t_* \circ u'_*$*, for every closed* t*-term* t_**.*

Proof. By inspection of the reduction rules, we see that, since we cannot have $u_* = \lambda x.u''_*; \lambda k.u'_*$, the statement holds for a one-step reduction. Then, by induction on the length of the reduction, we conclude. ☐

3.3 Readback

The ptq-calculus can be translated into the λ-calculus by a map that associates to each term of the ptq-calculus a typed λ-term. If we forget the types, every ptq-term is mapped into a λ-term that contains the same p-variables of the ptq-term and one special constant $\Box$, named *hole*, that plays the role of the free t-variable in the ptq-term.

A $\lambda_\Box$-term is a λ-term that may contain occurrences of the *hole* and whose variables range over the set of the p-variables. If M and N are $\lambda_\Box$-terms, the *hole composition* is defined by

$$M \circ N = M[N/\Box]$$

The hole composition is associative and $\Box$ is its neutral element, for $M \circ \Box = M = \Box \circ M$.

The (untyped) *readback* map $[\![\cdot]\!]$ is defined in Figure 3. Every t-term is mapped into a corresponding $\lambda_\Box$-term that may contain holes, while the other kind of terms are mapped into $\lambda_\Box$-terms that does not contain holes.

$$\begin{array}{lcllcl}
[\![*]\!] & = & \Box & [\![x]\!] & = & x \\
[\![\langle p, t_* \rangle]\!] & = & [\![t_*]\!] \circ \Box [\![p]\!] & [\![\lambda\langle x, k\rangle.u]\!] & = & \lambda x.[\![u_*]\!] \\
[\![\lambda x.u_*]\!] & = & [\![u_*]\!][\Box/x] & [\![\lambda k.u]\!] & = & [\![u_*]\!]
\end{array}$$

$$\begin{array}{lcl}
[\![\overline{\lambda} k.u]\!] & = & [\![u_*]\!] \\
[\![t_*; p]\!] & = & [\![t_*]\!] \circ [\![p]\!] \\
[\![q t_*]\!] & = & [\![t_*]\!] \circ [\![q]\!]
\end{array}$$

Figure 3: The readback map

The readback map naturally extends to judgments. The typed $\lambda_\Box$-terms obtained by the translation are terms of a typed λ-calculus $\Lambda_\Box$ in which:

- the set of the base types is the same of the ptq-calculus;
- the set of the variables is the set of the p-variables of the ptq-calculus;
- for each type A, there is a constant $\Box^A : A$, the *hole* of type A;
- the type assignment rules are those of the typed λ-calculus, with the restriction that
- a term of $\Lambda_\Box$ cannot contain occurrences of holes with different types (in any case, a term of $\Lambda_\Box$ may contain more than one hole of the same type);
- the reduction rule is the standard β-reduction.

Let $\Gamma = x_1 : A_1, \dots, x_n : A_n$ be a set of type assignments for variables. We shall denote by $\Gamma^p = x_1 : A_1^p, \dots, x_n : A_n^p$ the corresponding type assignment for p-variables. The readback of typing judgments is defined by:

$$\begin{array}{lcl}
[\![\Gamma^p \;\triangleright\; k : A^t \vdash t : B^t]\!] & = & \Gamma, \Box : B \vdash [\![t_*]\!] : A \\
[\![\Gamma^p \;\triangleright\; * : A^t \vdash t_* : B^t]\!] & = & \Gamma, \Box : B \vdash [\![t_*]\!] : A \\
[\![\Gamma^p \;\triangleright\; \vdash p : A^p]\!] & = & \Gamma \vdash [\![p]\!] : A \\
[\![\Gamma^p \;\triangleright\; \vdash q : A^q]\!] & = & \Gamma \vdash [\![q]\!] : A \\
[\![\Gamma^p \;\triangleright\; k : A^t \vdash u]\!] & = & \Gamma \vdash [\![u_*]\!] : A \\
[\![\Gamma^p \;\triangleright\; * : A^t \vdash u_*]\!] & = & \Gamma \vdash [\![u_*]\!] : A
\end{array}$$

Remark 5. Typed $\lambda_\Box$-terms with holes are a sort of typed *contexts*. But, differently from the standard definition of contexts, $\Lambda_\Box$-terms are equivalence classes modulo variable renaming (α-rule), for hole instantiation is not variable capturing. In a standard context, if a hole is in the scope of a λ-abstraction binding the variable x, the free occurrences of the variable x in a term M will be bound by the λ-abstraction when M is put into the hole. In a $\Lambda_\Box$-term, the hole composition $N \circ M = N[M/\Box]$ is defined by means of the standard variable substitution; therefore, the variable x bound in N must be renamed and the free occurrences of x in M remain free in $N \circ M$.

Proposition 6. *Let the judgment $\Gamma^p \;\triangleright\; \delta \vdash \xi$, be derivable in the* ptq*-calculus. The judgment $[\![\Gamma^p \;\triangleright\; \delta \vdash \xi]\!]$ is derivable in $\Lambda_\Box$.*

Proof. By induction on the derivation of $\Gamma^p \;\triangleright\; \delta \vdash \xi$. $\Box$

W.r.t. the readback, the $*$ plays the role of a neutral element, namely $[\![*; p]\!] = [\![p]\!]$ and $[\![q*]\!] = [\![q]\!]$. Indeed, the readback transformation maps the $*$-composition into the hole composition. (Let us recall that the $*$-composition on t-terms is defined by $t_* \circ t'_* = t'[t_*/k]$.)

Lemma 7.

1. $[\![t_* \circ t'_*]\!] = [\![t_*]\!] \circ [\![t'_*]\!]$

2. $[\![t_* \circ u_*]\!] = [\![t_*]\!] \circ [\![u_*]\!]$

Proof. By structural induction on t' and u.

1. Let $A = [\![t_* \circ t'_*]\!]$ and $B = [\![t_*]\!] \circ [\![t'_*]\!]$.

   (a) If $t' = k$, then $B = [\![t_*]\!] \circ [\![*]\!] = [\![t_*]\!] \circ \Box = [\![t_*]\!] = [\![t_*]\!] {*} \circ = A$.

   (b) If $t' = \langle t'', p\rangle$, then $A = [\![\langle t_* \circ t''_*, p\rangle]\!] =$ (by the definition of readback) $[\![t_* \circ t''_*]\!] \circ \Box [\![p]\!] =$ (by the induction hypothesis) $([\![t_*]\!] \circ [\![t''_*]\!]) \circ \Box [\![p]\!] = [\![t_*]\!] \circ ([\![t''_*]\!] \circ \Box [\![p]\!]) =$ (by the definition of readback) $[\![t_*]\!] \circ [\![\langle t''_*, p\rangle]\!] = B$.

   (c) If $t' = \lambda x.u$ (with $x \notin \mathsf{FV}(t)$), then $A = [\![t_* \circ \lambda x.u_*]\!] = [\![\lambda x.(t_* \circ u_*)]\!] =$ (by the definition of readback) $[\![t_* \circ u_*]\!][\Box/x] =$ (by the induction hypothesis) $([\![t_*]\!] \circ [\![u_*]\!])[\Box/x] =$ (by $x \notin \mathsf{FV}(t)$) $[\![t_*]\!] \circ ([\![u_*]\!][\Box/x]) =$ (by the definition of readback) $[\![t_*]\!] \circ [\![\lambda x.u_*]\!] = B$.

2. Let $A = [\![t_* \circ u_*]\!]$ and $B = [\![t_*]\!] \circ [\![u_*]\!]$.

   (a) If $u = t'; p$, then $A = [\![(t \circ t'_*); p]\!] =$ (by the definition of readback) $[\![t_* \circ t'_*]\!] \circ [\![p]\!] =$ (by the induction hypothesis) $([\![t_*]\!] \circ [\![t'_*]\!]) \circ [\![p]\!] = [\![t_*]\!] \circ ([\![t'_*]\!] \circ [\![p]\!]) =$ (by the definition of readback) $[\![t_*]\!] \circ [\![t'_*; p]\!] = B$.

   (b) If $u = qt'$, then $A = [\![q(t \circ t'_*)]\!] =$ (by the definition of readback) $[\![t_* \circ t'_*]\!] \circ [\![q]\!] =$ (by the induction hypothesis) $([\![t_*]\!] \circ [\![t'_*]\!]) \circ [\![q]\!] = [\![t_*]\!] \circ ([\![t'_*]\!] \circ [\![q]\!]) =$ (by the definition of readback) $[\![t_*]\!] \circ [\![qt'_*]\!] = B$.

□

Let us define $t_* \circ (t'_*; p) = (t_* \circ t'_*); p$ and $t_* \circ (qt'_*) = q(t_* \circ t'_*)$.

Corollary 8. $[\![t_* \circ u]\!] = [\![t_*]\!] \circ [\![u_*]\!]$

Proposition 10 proves that the readback is sound w.r.t. beta-reduction. In order to prove that proposition, we have to show (Lemma 9) that the readback is sound w.r.t. p-variable substitution.

Lemma 9. *Let a be a t-closed t-term or a p-term or a q-term or a t-closed e-term. Then*

$$[\![a[p/x]]\!] = [\![a]\!][[\![p]\!]/x]$$

Proof. By induction on the structure of a. □

Proposition 10. *If $u \to u'$, then $[\![u]\!] \xrightarrow{*} [\![u']\!]$. Moreover,*

1. *if $u = \langle p, t_* \rangle; \lambda\langle x, k \rangle.u'' \to u''[p/x, t_*/k] = u'$, then $[\![u]\!] =\!\!\to [\![u']\!]$;*

2. *otherwise, $[\![u]\!] = [\![u']\!]$.*

Proof. When $u = \langle p, t_* \rangle; \lambda\langle x, k \rangle.u'' \to u''[p/x, t_*/k]$, we have that

$$\begin{aligned}
&[\![\langle p, t_* \rangle; \lambda\langle x, k \rangle.u'']\!] \\
&\quad = [\![\langle p, t_* \rangle]\!] \circ [\![\lambda\langle x, k \rangle.u'']\!] = [\![\langle p, t_* \rangle]\!] \circ \lambda x.[\![u''_*]\!] = [\![t_*]\!] \circ (\lambda x.[\![u''_*]\!])[\![p]\!] \\
&\qquad \to [\![t_*]\!] \circ [\![u''_*]\!][[\![p]\!]/x] = \text{(by Lemma 9) } [\![t_*]\!] \circ [\![u''_*[p/x]]\!] \\
&\qquad\qquad = [\![t_* \circ u''_*[p/x]]\!] = [\![u''[p/x, t_*/k]]\!]
\end{aligned}$$

For the other reduction rules, we have instead

- $[\![*; \lambda k.u'']\!] = [\![u''_*]\!]$
- $[\![\langle p, t_* \rangle; \lambda k.u'']\!] = [\![\langle p, t_* \rangle]\!] \circ [\![u''_*]\!] = [\![\langle p, t_* \rangle \circ u''_*]\!] = [\![u''[\langle p, t* \rangle/k]]\!]$
- $[\![\lambda x.u''_*; p]\!] = [\![u''_*]\!][[\![p]\!]/x] = [\![u''[p/x]]\!]$

- $[\![(\overline{\lambda} k.u'')t_*]\!] = [\![t_*]\!] \circ [\![u''_*]\!] = [\![t_* \circ u''_*]\!] = [\![u''[t_*/k]]\!]$

$\square$

Concluding, we can state that:

- $\langle p, t_*\rangle;\lambda\langle x, k\rangle.u$ is a β-redex and that $\langle p, t_*\rangle;\lambda\langle x, k\rangle.u \to u'[p/x, t_*/k]$ is the β*-rule* of the ptq-calculus;
- all the other redexes of the calculus are *control redexes* and the corresponding rules are the *control rules* of the ptq-calculus;
- a reduction $u_* \overset{*}{\to} u'_*$ is a *control reduction* when it does not contract any β-redex.

3.4 Termination of the computations of the ptq-calculus

The reduction of any e-term u_* is deterministic—let us recall that any e-term is in normal form or is a redex (the only one in the term). Therefore, for every e-term there is only one maximal reduction sequence that, as we are going to prove, ends with a normal form.

By Proposition 10, any reduction of u_* cannot contain a number of β-rules greater than the length of the longest reduction of $[\![u_*]\!]$ (let us recall that such a term is typable in the simply typed λ-calculus, thus it is strongly normalizing). Therefore, if the e-term u_* has an infinite reduction, such a reduction must eventually end in an infinite sequence of control reductions.

In order to prove that the control reduction are terminating, we can associate a measure to every (t-closed) ptq-term that, given a function from the set of the p-variables into the set of the natural numbers $\mathbb{N}$, maps

- every t-term into a function of type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$;
- every p-term into a natural number;
- every q-term and every e-term into a function of type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$.

Denoting by $|a|_\sigma$ the map that, given a function σ from the set of the p-variables into $\mathbb{N}$, associates to a ptq-term a its measure, the maps

$$|t_*|_\sigma : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N} \qquad |p|_\sigma : \mathbb{N}$$

$$|q|_\sigma : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \qquad |u_*|_\sigma : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$

are defined by

$$\begin{array}{lcl@{\qquad}lcl}
|*|_\sigma\, f\, n & = & f\, n & |x|_\sigma & = & \sigma(x) \\
|\langle p, t_*\rangle|_\sigma\, f\, n & = & n & |\lambda\langle x, k\rangle.u|_\sigma & = & 0 \\
|\lambda x.u_*|_\sigma\, f\, n & = & |u_*|_{\sigma[x\mapsto n]}(f) & |\lambda k.u|_\sigma & = & |u_*|_\sigma\, \mathsf{id}
\end{array}$$

$$\begin{array}{rcl}|\overline{\lambda} k.u|_\sigma\, f &=& |u_*|_\sigma\, f\\ |t_*;p|_\sigma\, f &=& (|t_*|_\sigma\, f\,|p|_\sigma)+1\\ |qt_*|_\sigma\, f &=& (|q|_\sigma\,(|t_*|_\sigma\, f))+1\end{array}$$

The key properties that have inspired the definition of the above measure are summarized by the following fact.

Fact 11. *Let a be a* t*-closed* t*-term or a* p*-term or a* q*-term or a* t*-closed* e*-term.*

1. $|a[p/x]|_\sigma = |a|_{\sigma[x\mapsto |p|_\sigma]}$*, for every* p*-term* p*.*
2. $|t_* \circ a|_\sigma\, f = |a|_\sigma(|t_*|_\sigma\, f)$*, for every* t*-closed* t*-term* t_**, when* a *is a* t*-closed* t*-term or a* t*-closed* e*-term.*

Proof. By induction on the structure of a.

1. Straightforward.
2. In the base case $a = *$, we have $|t_* \circ *|_\sigma\, f = |t_*|_\sigma\, f = |*|_\sigma\,(|t_*|_\sigma\, f)$. The induction steps are:

   (a) if $a = \langle p, t'_*\rangle$, then $|t_* \circ \langle p, t'_*\rangle|_\sigma\, f = \mathsf{id} = |\langle p, t'_*\rangle|_\sigma\,(|t_*|_\sigma\, f)$;

   (b) if $a = \lambda x.u_*$ with $x \notin \mathsf{FV}(t_*)$, then $|t_* \circ \lambda x.u_*|_\sigma\, f = \lambda n.|t_* \circ u_*|_{\sigma[x\mapsto n]}\, f$ $=$ (by the induction hypothesis and $x \notin \mathsf{FV}(t_*)$) $\lambda n.|u_*|_{\sigma[x\mapsto n]}\,(|t_*|_\sigma\, f)$ $= |\lambda x.u_*|_\sigma\,(|t_*|_\sigma\, f)$;

   (c) if $a = t'_*;p$, then $|t_* \circ (t'_*;p)|_\sigma\, f = |t_* \circ t'_*|_\sigma\, f\,|p_*|_\sigma + 1 =$ (by the induction hypothesis) $|t'_*|_\sigma\,(|t_*|_\sigma\, f)\,|p_*|_\sigma + 1 = |t'_*;p|_\sigma\,(|t_*|_\sigma\, f)$;

   (d) if $a = qt'_*$, then $|t_* \circ (qt'_*)|_\sigma\, f = |q|_\sigma\,(|t_* \circ t'_*|_\sigma\, f) + 1 =$ (by the induction hypothesis) $|q|_\sigma(|t'_*|_\sigma\,(|t_*|_\sigma\, f)) + 1 = |qt'_*|_\sigma\,(|t_*|_\sigma\, f)$.

□

The measure of an e-term u_* corresponds to the length of its longest control reduction. Let us define

$$\ell(u_*) = \sup\{l \mid l \text{ is the length of a control reduction } u_* \xrightarrow{*} u'_*\}$$

.

Lemma 12. *For every* e*-term,* $\ell(u_*)$ *is finite. Moreover,* $\ell(u_*) = |u_*|_o\,\mathsf{id} - 1$*, where* o *is the map that associates* 0 *to every free variable of* u_**.*

Proof. Let us prove by induction on $\ell(u_*)$ that $|u_*|_o\,\mathsf{id} = \ell(u_*) + 1$. We proceed by case analysis.

1. Let $u_* = t_*;p$ with $t_* = *$ or $t_* = \langle p', t'_*\rangle$. We have that $|t_*;p|_o\,\mathsf{id} = |t_*|_o\,\mathsf{id}\,|p|_o + 1 = |p|_o + 1$. Therefore, we have to show that $|p|_o = \ell(t_*;p)$.

   (a) if $p = x$, then $|x|_o = o(x) = 0 = \ell(t_*;x)$, since $t_*;x$ is a normal form;

(b) if $p = \lambda\langle k, x\rangle.u'_*$, then $|\lambda\langle k, x\rangle.u'_*|_o = 0 = \ell(t_*; \lambda\langle k, x\rangle.u'_*)$ since $t_*; \lambda\langle k, x\rangle.u'_*$ is a normal form for the control rules;

(c) if $p = \lambda k.u'_*$, then $t_*; \lambda k.u'_* \rightarrow t_* \circ u'_*$ and $|\lambda k.u'_*|_o = |u'_*|_o\, \mathsf{id} =$ (since $|t_*|_o\, \mathsf{id} = \mathsf{id}$, by the hypothesis on t_*) $|u'_*|_o\, (|t_*|_o\, \mathsf{id}) =$ (by Fact 11) $|t_* \circ u'_*|_o\, \mathsf{id} =$ (by the induction hypothesis) $\ell(t_* \circ u'_*) + 1 = \ell(t_*; \lambda k.u'_*)$.

2. Let $u_* = \lambda x.u'_*; p$. We have that $\lambda x.u'_*; p \rightarrow u'_*[p/x]$ and $|\lambda x.u'_*; p|_o\, \mathsf{id} = |\lambda x.u'_*|_o\, \mathsf{id}\, |p|_o + 1 = |u'_*|_{o[x \mapsto |p|_o]}\, \mathsf{id} + 1 =$ (by Fact 11) $|u'_*[p/x]|_o\, \mathsf{id} + 1 =$ (by the induction hypothesis) $\ell(u'_*[p/x]) + 2 = \ell(\lambda x.u'_*; p) + 1$.

3. Let $u_* = (\overline{\lambda} k.u'_*)t_*$. We have that $(\overline{\lambda} k.u')t_* \rightarrow t_* \circ u'_*$ and $|(\overline{\lambda} k.u')t_*|_o\, \mathsf{id} = |\overline{\lambda} k.u'|_o\, (|t_*|_o\, \mathsf{id}) + 1 = |u'_*|_o\, (|t_*|_o\, \mathsf{id}) + 1 =$ (by Fact 11) $|t_* \circ u'_*|_o\, \mathsf{id} + 1 =$ (by the induction hypothesis) $\ell(t_* \circ u'_*) + 2 = \ell((\overline{\lambda} k.u'_*)t_*) + 1$.

□

We can then conclude that the ptq-calculus is (strongly) normalizing.

Theorem 13. *There is no infinite reduction of any* e*-term of the* ptq*-calculus.*

Proof. By Proposition 10, the maximal reduction of an e-term u_* cannot contain an infinite number of β-rules. By Lemma 12, that reduction cannot contain an infinite sequence of control rules neither. □

The previous result ensures that the ptq-calculus may be used as a computational tool for the implementation of β-reduction. In fact, Theorem 13 implies that, when the t-closed e-term u_* is a representation of a given simply typed λ-term M, the reduction of u_* terminates with a representation of the normal form of M.

Theorem 14. *Let N be the normal form of a simply typed λ-term M. If u_* is an* e*-term s.t. $[\![u_*]\!] = M$, there is $u_* \overset{*}{\mapsto} u'_*$ s.t. $[\![u'_*]\!] = N$. In particular, $[\![u'_*]\!] = N$ when u'_* is the normal form of u_*.*

Proof. By Proposition 10 and Theorem 13. □

4 Translations

Theorem 14 shows that the ptq-calculus is a well-suited target language for the "compilation" of λ-terms: given a suitable translation of λ-terms into ptq-terms, we can compute the normal form of a λ-term by reducing its corresponding ptq-term; where by suitable translation we mean a (total) map that inverts the readback.

Let us remind that, since in a ptq-term there is at most one redex, the reduction of the ptq-term is deterministic and induces a particular reduction strategy of its readback. As a consequence, any translation of λ-terms into ptq-terms defines a reduction strategy for λ-terms. In particular, and this is the interesting computational property of the ptq-calculus, we can define translations that implement Call-by-Value and Call-by-Name (see Plotkin [14]).

4.1 Call-by-Value and Call-by-Name λ-calculus

In the Call-by-Value (CbV) and in the Call-by-Name (CbN) λ-calculus a λ-term is a *value* if it is not an application. The main rule, in both cases, is the β-rule of λ-calculus that, in CbV, is restricted to the case in which a λ-abstraction is applied to a value.

In the paper we shall consider the lazy case only, that is we shall not reduce the β-redexes that are in the scope of a λ-abstraction.

The reduction rules of CbN and CbV will be given by means of inference rules that do not extend to contexts. In the case of CbN, we have two rules (small step natural semantics of CbN):

$$\frac{}{(\lambda x.M)N \mapsto_n M[N/x]}\,\beta_n \qquad \frac{M \mapsto_n M_1}{MN \mapsto_n M_1 N}$$

The β_n-rule is the standard β-reduction of λ-calculus restricted to the case in which the reducing term is a β-redex. The second rule, instead, allows to reduce a β-redex when it is the left-most-outer-most head application of the term.

The CbN-normal form of any closed λ-term M is a value V, say $M \Downarrow_n V$. The relation $\Downarrow_n$ is defined by the following rules (big step natural semantics of CbN):

$$\frac{}{V \Downarrow_n V} \qquad \frac{M \Downarrow_n \lambda x.M_1 \quad M_1[N/x] \Downarrow_n V}{MN \Downarrow_n V}$$

where V denotes a value.

The main reduction rule of CbV is the usual β-rule restricted to the case in which the reducing term is a β-redex whose argument is a value, namely

$$\frac{}{(\lambda x.M)V \mapsto_v M[V/x]}\,\beta_v$$

where V denotes a value.

As in the case of CbN, the small steps natural semantics of CbV is completed by the inference rules that allow to reduce the head redexes of an application that, in this case, can be in the argument part also. However, we have to fix an evaluation order deciding which part of an application we want to reduce first. The rules that reduce the function part first are

$$\frac{M \mapsto_v M_1}{MN \mapsto_v M_1 N} \qquad \frac{N \mapsto_v N_1}{VN \mapsto_v VN_1}$$

where V denotes a value. Otherwise, if one wants to reduce the argument first, the rules are

$$\frac{N \mapsto_v N_1}{MN \mapsto_v MN_1} \qquad \frac{M \mapsto_v M_1}{MV \mapsto_v M_1 V}$$

where V denotes a value.

Both choices lead to the following big step natural semantics for CbV

$$\frac{}{V \Downarrow_n V} \qquad \frac{M \Downarrow_v \lambda x.M_1 \quad M_1[W/x] \Downarrow_v V \quad N \Downarrow_v W}{MN \Downarrow_n V}$$

where V and W denote values.

The typing rules of CbV and CbN are the usual ones of simply typed λ-calculus.

4.2 Plotkin's translations

In his seminal paper [14], Plotkin gave two translations that allow to implement CbN by CbV and vice versa. Both the translations map an application into a value, that is into a term that is in normal form—let us remind that we do not reduce in the scope of an abstraction. In order to start the computation of the translated term, we have to pass the identity $\mathrm{I} = \lambda x.x$ to it—the term I plays the role of the initial continuation.

4.2.1 CbN translation

The CbN translation is defined by the map

$$\begin{array}{rcl} \lceil x \rceil^n & = & x \\ \lceil \lambda x.M \rceil^n & = & \lambda k.k(\lambda x.\lceil M \rceil^n) \\ \lceil MN \rceil^n & = & \lambda k.\lceil M \rceil^n(\lambda m.m\lceil N \rceil^n k) \end{array}$$

that translates a λ-term M into another λ-term $\lceil M \rceil^n$ s.t. the CbV reduction of $\lceil M \rceil^n$ corresponds to the CbN reduction of $M\mathrm{I}$.

The untyped CbN translation of terms given above corresponds to the following translation of typed terms

$$\Gamma \vdash M : A \quad \overset{\text{CbN}}{\rightsquigarrow} \quad \lceil \Gamma \rceil^n \vdash \lceil M \rceil^n : \lceil A \rceil^n$$

where, if $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, then $\lceil \Gamma \rceil^n = x_1 : \lceil A_1 \rceil^n, \ldots, x_n : \lceil A_n \rceil^n$, and the translation of the types $\lceil A \rceil^n$ is defined by

$$\lceil A \rceil^n \;=\; \lfloor A \rfloor_n \to o \to o$$
$$\lfloor X \rfloor_n \;=\; X \qquad\qquad \lfloor A \to B \rfloor_n \;=\; \lceil A \rceil^n \to \lceil B \rceil^n$$

where o is a new base type and X denotes a base type.

For the analysis of the correspondence between Plotkin's translation and the ptq-translation that we shall give in the following, let us observe that, by uncurryfying the translation of $A \to B$, we can assume

$$\lfloor A \to B \rfloor_n \quad = \quad (\lceil A \rceil^n \times (\lfloor B \rfloor_n \to o)) \to o$$

Correspondingly, the CbN translation of terms becomes

$$\begin{aligned}\lceil x\rceil^n &= x\\ \lceil \lambda x.M\rceil^n &= \lambda k.k(\lambda(x,h).\lceil M\rceil^n h)\\ \lceil MN\rceil^n &= \lambda k.\lceil M\rceil^n(\lambda m.m(\lceil N\rceil^n,k))\end{aligned}$$

4.2.2 CbV translation

The CbV translation is defined by the map

$$\begin{aligned}\lceil x\rceil^v &= \lambda k.kx\\ \lceil \lambda x.M\rceil^v &= \lambda k.k(\lambda x.\lceil M\rceil^v)\\ \lceil MN\rceil^v &= \lambda k.\lceil M\rceil^v(\lambda m.\lceil N\rceil^v(\lambda n.mnk))\end{aligned}$$

that translates every λ-term M into another λ-term $\lceil M\rceil^v$ s.t. the CbN reduction of $\lceil M\rceil^v$ corresponds to the CbV reduction of MI.

The map above ensures that, in an application MN, the function M is evaluated first. Replacing the translation of MN with

$$\lceil MN\rceil^v = \lambda k.\lceil N\rceil^v(\lambda n.\lceil M\rceil^v(\lambda m.mnk))$$

we get the translation for the case in which, in an application MN, the argument N is evaluated first.

The untyped CbV translation of terms given above corresponds to the following translation of typed terms

$$\Gamma\vdash M:A \quad \overset{\text{CbV}}{\rightsquigarrow} \quad \lfloor\Gamma\rfloor_v\vdash\lceil M\rceil^v:\lceil A\rceil^v$$

where, if $\Gamma = x_1 : A_1,\dots,x_n : A_n$, then $\lfloor\Gamma\rfloor_v = x_1 : \lfloor A_1\rfloor_v,\dots,x_n : \lfloor A_n\rfloor_v$ and the translations of types $\lfloor A\rfloor_v$ and $\lceil A\rceil^v$ are defined by

$$\lceil A\rceil^v = \lfloor A\rfloor_v \to o\to o$$
$$\lfloor X\rfloor_n = X \qquad\qquad \lfloor A\to B\rfloor_n = \lfloor A\rfloor_v\to\lceil B\rceil^v$$

where o is a new base type and X denotes a base type.

As in the case of CbN translation, by uncurryfying the translation of $A\to B$, we get

$$\lfloor A\to B\rfloor_n = (\lfloor A\rfloor_v\times(\lfloor B\rfloor_v\to o))\to o$$

Correspondingly, the CbV translation of terms becomes

$$\begin{aligned}\lceil x\rceil^v &= \lambda k.kx\\ \lceil \lambda x.M\rceil^v &= \lambda k.k(\lambda(x,h).\lceil M\rceil^v h)\\ \lceil MN\rceil^v &= \begin{cases}\lambda k.\lceil M\rceil^v(\lambda m.\lceil N\rceil^v(\lambda n.m(n,k))) & \text{eval } M \text{ first}\\ \lambda k.\lceil N\rceil^v(\lambda n.\lceil M\rceil^v(\lambda m.m(n,k))) & \text{eval } N \text{ first}\end{cases}\end{aligned}$$

where, for MN, we have to choose the upper translation if we want to evaluate M first or the lower translation if we want to evaluate N first.

Call-by-Name			Call-by-Value		
$\overline{x}^n$	$=$	x	$\overline{x}^v$	$=$	$\overline{\lambda} k.k;x$
$\overline{\lambda x.M}^n$	$=$	$\lambda\langle x,k\rangle.k;\overline{M}^n$	$\overline{\lambda x.M}^v$	$=$	$\overline{\lambda} k.k;(\lambda\langle x,k\rangle.\overline{M}^v k)$
$\overline{MN}^n$	$=$	$\lambda k.\langle\overline{N}^n,k\rangle;\overline{M}^n$	$\overline{MN}^v$	$=$	$\overline{\lambda} k.\overline{N}^v(\lambda x.\overline{M}^v\langle x,k\rangle)$

Figure 4: ptq-translations

4.3 Call-by-Name ptq-translation

The CbN ptq-translation derives from the translation that maps every sequent derivable in minimal logic into the corresponding ptq-sequent, by translating every formula of minimal logic into a p-formula, that is

$$\Gamma \vdash A \quad \overset{\text{CbN}}{\leadsto} \quad \Gamma^p \vdash A^p$$

The simplest way to get the above correspondence is by the CbN-translation in Figure 4.

Proposition 15. *Let* $\Gamma \vdash M : A$ *be derivable in the simply typed* λ*-calculus. Then,* $\Gamma^p \vdash \overline{M}^n : A^p$ *is derivable in the* ptq*-calculus.*

Proof. By induction on the structure of M. □

4.3.1 Correspondence with Plotkin's CbN translation

In order to relate the CbN ptq-translation with Plotkin's CBN translation, let us observe that we may map the translated Plotkin's types according to the following schema:

$$\lceil A\rceil^n \quad = \quad \lfloor A\rfloor_n \to o \to o \quad \leadsto \quad A^p$$

$$\lfloor A\rfloor_n \quad \leadsto \quad A^p \qquad\qquad \lfloor A\rfloor_n \to o \quad \leadsto \quad A^t$$

that also implies, in the translation with pairs,

$$\lfloor A\to B\rfloor_n \to o \quad = \quad \lceil A\rceil^n \times (\lfloor B\rfloor_n \to o) \quad \leadsto \quad A\to B^t$$

Correspondingly, the CbN Plotkin's translation of terms becomes

$$\begin{array}{rcl} \lceil x\rceil^n & \leadsto & x \\ \lceil \lambda x.M\rceil^n & \leadsto & \lambda k.k;(\lambda(x,m).m;\overline{M}^n) \\ \lceil MN\rceil^n & \leadsto & \lambda k.(\lambda m.\langle\overline{N}^n,k\rangle;m);\overline{M}^n \end{array}$$

Such a translation can be simplified to the ptq-translation in Figure 4 by observing that, by η-equivalence $\lambda k.(\lambda m.\langle\lceil N\rceil^n,k\rangle;m);\lceil M\rceil^n = \lambda k.\langle\lceil N\rceil^n,k\rangle;\lceil M\rceil^n$ and that, since in the translation we never use t-terms with the shape $\lambda x.u$, we can also assume that $\lambda k.k;(\lambda\langle x,m\rangle.m;\lceil M\rceil^n)$ is equivalent to $\lambda\langle x,m\rangle.m;\lceil M\rceil^n$.

4.4 Call-by-Value ptq-translation

The CbV ptq-translation derives from the translation of minimal logic that maps the premises of the ending sequent into p-formulas and its conclusion into a q-formula

$$\Gamma \vdash A \qquad \rightsquigarrow \qquad \Gamma^{p} \vdash A^{q}$$

In order to get the above correspondence, one can easily find the CbV translation in Figure 4.

Proposition 16. *Let* $\Gamma \vdash M : A$ *be derivable in the simply typed* λ*-calculus. Then,* $\Gamma^{p} \vdash \overline{M}^{v} : A^{q}$ *is derivable in the* ptq*-calculus.*

Proof. By induction on the structure of M. $\square$

4.4.1 Correspondence with Plotkin's CbV translation

In order to relate the CbV ptq-translation with Plotkin's CbV translation, the translated Plotkin's types can be mapped to ptq-types according to the following schema:

$$\lceil A \rceil^{n} \quad = \quad \lfloor A \rfloor_{n} \to o \to o \quad \rightsquigarrow \quad A^{q}$$

$$\lfloor A \rfloor_{n} \quad \rightsquigarrow \quad A^{p} \qquad\qquad \lfloor A \rfloor_{n} \to o \quad \rightsquigarrow \quad A^{t}$$

that also implies, in the translation with pairs,

$$\lfloor A \to B \rfloor_{n} \to o \quad = \quad \lfloor A \rfloor_{n} \times (\lfloor B \rfloor_{n} \to o) \quad \rightsquigarrow \quad A \to B^{t}$$

Correspondingly, the CbV Plotkin's translation of terms becomes

$$\begin{array}{rcl}
\lceil x \rceil^{v} & \rightsquigarrow & \overline{\lambda} k.k; x \\
\lceil \lambda x.M \rceil^{v} & \rightsquigarrow & \overline{\lambda} k.k; (\lambda \langle x, h \rangle.h; \overline{M}^{v}) \\
\lceil MN \rceil^{v} & \rightsquigarrow & \left\{ \begin{array}{l} \overline{\lambda} k.\overline{M}^{v}(\lambda m.(\overline{N}^{v}(\lambda n.\langle n, k \rangle; m))) \\ \overline{\lambda} k.\overline{N}^{v}(\lambda n.(\overline{M}^{v}(\lambda m.\langle n, k \rangle; m))) \end{array} \right.
\end{array}$$

The translation in which M is evaluated first cannot be simplified. The case in which N is evaluated first, instead, can be reduced to the CbV ptq-translation in Figure 4 because of the η-equivalence $\lambda m.\langle n, k \rangle; m = \langle n, k \rangle$.

5 Properties of the ptq-translations

5.1 Precomputation

The ptq-translations map λ-terms into p-terms that, in the ptq-calculus, are irreducible. In order to eval a translated term we have to combine it with a test. The natural choice for such an initial test is the constant $*$ that, as already remarked, in our framework plays the role of the initial continuation.

By reducing the ptq-translation u_{*} of a λ-term M, we do not get the translation u'_{*} of some reduct N of M. We shall see instead that we can get some

u''_* s.t. $[\![u''_*]\!] = N$, which differs from u'_* for the reduction of some control redexes, namely $u'_* \xrightarrow{*} u''_*$ by a sequence of control rules. Since we know that the readback $M = [\![u_*]\!]$ of an e-term u_* is not changed by the control rules (Proposition 10) and that the control reductions are terminating (Lemma 12), we can take the normal form of u_* for the control rules as another standard representation of M. The computation of such a normal form can be merged with the ptq-translations by defining two variants of the ptq-translations that map λ-terms into e-terms.

The Call-by-Name translation from λ-terms to e-terms is defined by:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\overline{\overline{V}}^n & = & *;\underline{V}_n & \underline{x}_n & = & x \\
\overline{\overline{MN}}^n & = & \langle \overline{N}^n, * \rangle \circ \overline{\overline{M}}^n & \underline{\lambda x.M}_n & = & \lambda\langle x, k\rangle.k;\overline{M}^n
\end{array}$$

where V is a value. The Call-by-Value translation from λ-terms to e-terms is defined by:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\overline{\overline{V}}^v & = & *;\underline{V}_v & \underline{x}_v & = & x \\
\overline{\overline{MV}}^v & = & \langle \underline{V}_v, * \rangle \circ \overline{\overline{M}}^v & \underline{\lambda x.M}_v & = & \lambda\langle x, k\rangle.\overline{M}^v k \\
\overline{\overline{MN}}^v & = & (\lambda x.\overline{M}^v \langle x, *\rangle) \circ \overline{\overline{N}}^v & & &
\end{array}$$

where V is a value and N is not a value.

For every value V,

$$\overline{V}^n = \lambda k.k;\underline{V}_n \qquad\qquad \overline{V}^v = \overline{\lambda} k.k;\underline{V}_v$$

Remark 17. Let us define

$$[p_1, \ldots, p_k] \quad = \quad \begin{cases} \langle \overline{p_1}^n, \langle \ldots, \langle \overline{p_k}^n, * \rangle \ldots \rangle\rangle & \text{for } k > 0 \\ * & \text{for } k = 0 \end{cases}$$

1. $\overline{\overline{M\,M_1 \ldots M_k}}^n = [\overline{M_1}^n, \ldots, \overline{M_k}^n] \circ \underline{M}_n$. In particular, when $M = V$ is a value
$$\overline{\overline{V\,M_1 \ldots M_k}}^n \quad = \quad [\overline{M_1}^n, \ldots, \overline{M_k}^n];\underline{V}_n$$
2. $\overline{\overline{M\,V_1 \ldots V_k}}^v = [\overline{V_1}^v, \ldots, \overline{V_k}^v] \circ \underline{M}_v$where $V_1, \ldots, V_k$ are values. In particular, when $M = V$ is a value and when $M = PQ$ where Q is not a value
$$\begin{array}{rcl}
\overline{\overline{V\,V_1 \ldots V_k}}^v & = & [\underline{V_1}_v, \ldots, \underline{V_k}_v];\underline{V}_v \\
\overline{\overline{PQV_1 \ldots V_k}}^v & = & (\lambda x.\overline{P}^v[x, \underline{V_1}_v, \ldots, \underline{V_k}_v]) \circ \overline{\overline{Q}}^v
\end{array}$$

Lemma 18. *For every* λ*-term* M*,*

1. both $\overline{\overline{M}}^n$ *and* $\overline{\overline{M}}^v$ *are in normal form for the control rules;*

2. *by a sequence of control rules*

   (a) $*;\overline{\mathsf{M}}^{n} \xrightarrow{*} \overline{\overline{\mathsf{M}}}^{n}$,

   (b) $\overline{\mathsf{M}}^{\nu} * \xrightarrow{*} \overline{\overline{\mathsf{M}}}^{\nu}$.

Proof. Let us separately prove the two items of the statement.

1. Let us start by proving the following claim.

   Claim . There are two values V^n and V^ν and two t-closed terms t^n_* and t^ν_* s.t. $\overline{\overline{\mathsf{M}}}^{n} = \mathsf{t}^n_*;\underline{\mathsf{V}^n}_n$ and $\overline{\overline{\mathsf{M}}}^{\nu} = \mathsf{t}^\nu_*;\underline{\mathsf{V}^\nu}_\nu$, and s.t.:

   (a) when M is a value, $\mathsf{t}^n_* = \mathsf{t}^\nu_* = *$ and $\mathsf{M} = \mathsf{V}^n = \mathsf{V}^\nu$;

   (b) when M is not a value, t^n_* and t^ν_* are pairs, namely $\mathsf{t}^n_* = \langle \mathsf{p}', \mathsf{t}'_* \rangle$ for some p' and t'_* and $\mathsf{t}^\nu_* = \langle \mathsf{p}'', \mathsf{t}''_* \rangle$ for some p'' and t''_*.

   Proof of the claim. For $\overline{\overline{\mathsf{M}}}^{n}$, the proof immediately follows by item 1 of Remark 17: take $\mathsf{M} = \mathsf{V}^n\,\mathsf{M}_1 \ldots \mathsf{M}_k$. For $\overline{\overline{\mathsf{M}}}^{\nu}$, the proof exploits the inductive definition of $\overline{\overline{\mathsf{M}}}^{\nu}$ in item 2 of Remark 17: the base case is immediate, just take $\mathsf{M} = \mathsf{V}^\nu\,\mathsf{V}_1 \ldots \mathsf{V}_K$; the induction step $\mathsf{M} = \mathsf{P}\mathsf{Q}\mathsf{V}_1 \ldots \mathsf{V}_k$ holds by the induction hypothesis on $\overline{\overline{\mathsf{Q}}}^{\nu}$ and by the fact that Q is not a value. □

   Then, in order to conclude that $\overline{\overline{\mathsf{M}}}^{n} = \mathsf{t}^n_*;\underline{\mathsf{V}^n}_n$ and $\overline{\overline{\mathsf{M}}}^{\nu} = \mathsf{t}^\nu_*;\underline{\mathsf{V}^\nu}_\nu$ are in normal form for the control rules, let us observe that, for any value V:

   (a) if $\mathsf{V} = \mathsf{x}$, then $\underline{\mathsf{V}}_n = \underline{\mathsf{V}}_\nu = \mathsf{x}$;

   (b) if $\mathsf{V} = \lambda \mathsf{x}.\mathsf{N}$, then $\underline{\mathsf{V}}_n = \lambda\langle \mathsf{x}, \mathsf{k}\rangle.\mathsf{u}^n$ and $\underline{\mathsf{V}}_\nu = \lambda\langle \mathsf{x}, \mathsf{k}\rangle.\mathsf{u}^\nu$ for some u^n and u^ν.

2. By structural induction on M. When $\mathsf{M} = \mathsf{V}$, where V is a value (base case), it is readily seen that $*;\overline{\mathsf{V}}^{n} \longrightarrow *;\underline{\mathsf{V}}_n$ and $\overline{\mathsf{V}}^{\nu} * \longrightarrow *;\underline{\mathsf{V}}_\nu$ by a control reduction. When $\mathsf{M} = \mathsf{P}\mathsf{Q}$, by the induction hypothesis, we have three control reductions s.t.: $*;\mathsf{P} \xrightarrow{*} \overline{\overline{\mathsf{P}}}^{n}$, for the CbN; $\overline{\mathsf{P}}^{\nu} * \xrightarrow{*} \overline{\overline{\mathsf{P}}}^{\nu}$ and $\overline{\mathsf{Q}}^{\nu} * \xrightarrow{*} \overline{\overline{\mathsf{Q}}}^{\nu}$, for the CbV. Therefore, by Lemma 4,

   - for the CbN, we have the control reduction
     $*;\overline{\mathsf{M}}^{n} = *;\overline{\mathsf{P}\mathsf{Q}}^{n} \longrightarrow \langle \overline{\mathsf{Q}}^{n}, * \rangle;\overline{\mathsf{P}}^{n} \xrightarrow{*} \langle \overline{\mathsf{Q}}^{n}, * \rangle \circ \overline{\overline{\mathsf{P}}}^{n} = \overline{\overline{\mathsf{M}}}^{n}$;
   - for the CbV,

     (a) when $\mathsf{Q} = \mathsf{V}$ is a value, we have the control reduction
     $*;\overline{\mathsf{M}}^{\nu} = \overline{\mathsf{P}\mathsf{V}}^{\nu} * = \langle \underline{\mathsf{V}}_\nu, * \rangle \circ \overline{\mathsf{P}}^{\nu} \xrightarrow{*} \langle \underline{\mathsf{V}}_\nu, * \rangle \circ \overline{\overline{\mathsf{P}}}^{\nu} = \overline{\overline{\mathsf{M}}}^{\nu}$;

     (b) when Q is not a value, we have the control reduction
     $*;\overline{\mathsf{M}}^{\nu} = \overline{\mathsf{P}\mathsf{Q}}^{\nu} * \longrightarrow (\lambda \mathsf{x}.\overline{\mathsf{P}}^{\nu}\langle \mathsf{x}, * \rangle) \circ \overline{\mathsf{Q}}^{\nu} \xrightarrow{*} (\lambda \mathsf{x}.\overline{\mathsf{P}}^{\nu}\langle \mathsf{x}, * \rangle) \circ \overline{\overline{\mathsf{Q}}}^{\nu} = \overline{\overline{\mathsf{M}}}^{\nu}$.

□

5.2 Readback

One of the key properties of the ptq-translations is that the readback of a translated term is the term itself.

Proposition 19. *For every* λ*-term* M*,*

1. $[\![*;\overline{M}^n]\!] = [\![\overline{\overline{M}}^n]\!] = [\![\overline{M}^n]\!] = M$
2. $[\![\overline{M}^v *]\!] = [\![\overline{\overline{M}}^v]\!] = [\![\overline{M}^v]\!] = M$

Proof. By the definition of readback, it is readily seen that $[\![*;\overline{M}^n]\!] = [\![\overline{M}^n]\!]$ and that $[\![\overline{M}^v *]\!] = [\![\overline{M}^v]\!]$.

1. We shall prove $[\![\overline{M}^n]\!] = M$, by induction on the structure of M.
   (a) $[\![\overline{x}^n]\!] = x$
   (b) $[\![\overline{\lambda x.P}^n]\!] = [\![\lambda\langle x,k\rangle.k;\overline{P}^n]\!] = \lambda x.[\![*;\overline{P}^n]\!] =$ (by the induction hypothesis) $\lambda x.P$
   (c) $[\![\overline{PQ}^n]\!] = [\![\lambda k.\langle\overline{Q}^n,k\rangle;\overline{P}^n]\!] = [\![\langle\overline{Q}^n,*\rangle]\!] \circ [\![\overline{P}^n]\!] = [\![\overline{P}^n]\!][\![\overline{Q}^n]\!] =$ (by the induction hypothesis) PQ

2. By induction on the structure of M, we shall prove that $[\![\overline{M}^v t_*]\!] = [\![t_*]\!] \circ M$.
   (a) $[\![\overline{x}^v t_*]\!] = [\![t_*]\!] \circ [\![\overline{\lambda} k.k;x]\!] = [\![t_*]\!] \circ [\![*;x]\!] = [\![t_*]\!] \circ x$
   (b) $[\![\overline{PQ}^v t_*]\!] = [\![t_*]\!] \circ [\![\overline{Q}^v(\lambda x.\overline{P}^v\langle x,*\rangle)]\!] =$ (by the induction hypothesis) $[\![t_*]\!] \circ [\![\lambda x.\overline{P}^v\langle x,*\rangle]\!] \circ Q = [\![t_*]\!] \circ [\![\overline{P}^v\langle x,*\rangle]\!][Q/x] =$ (by the induction hypothesis) $[\![t_*]\!] \circ ([\![\langle x,*\rangle]\!] \circ P)[Q/x] = [\![t_*]\!] \circ Px[Q/x] = [\![t_*]\!] \circ PQ$
   (c) $[\![\overline{\lambda x.P}^v t_*]\!] = [\![t_*]\!] \circ [\![\lambda\langle x,k\rangle.\overline{P}^v k]\!] = [\![t_*]\!] \circ \lambda x.[\![\overline{P}^v *]\!] =$ (by the induction hypothesis) $[\![t_*]\!] \circ \lambda x.P$

   In particular, $[\![\overline{M}^v *]\!] = [\![*]\!] \circ M = M$.

By Lemma 18, $*;\overline{M}^n \xrightarrow{*} \overline{\overline{M}}^n$ and $\overline{M}^v * \xrightarrow{*} \overline{\overline{M}}^v$ by control reductions. Therefore, by Proposition 10, $[\![\overline{\overline{M}}^n]\!] = [\![*;\overline{M}^n]\!] = M$ and $[\![\overline{\overline{M}}^v]\!] = [\![\overline{M}^v *]\!] = M$. □

5.3 Soundness and Completeness

Lemma 20. *For every pair of* λ*-terms* M, N *and every value* V

1. $\overline{M[N/x]}^n = \overline{M}^n[\overline{N}^n/x]$
2. $\overline{M[V/x]}^n = \overline{M}^n[\underline{V}_v/x]$

Proof. By induction on M. □

Proposition 21. *For every* λ*-term* M*.*

1. $\mathsf{M} \to \mathsf{N}$ *in the CbN* λ*-calculus iff* $\overline{\overline{\mathsf{M}}}^{n} \xrightarrow{*} \overline{\overline{\mathsf{N}}}^{n}$.

2. $\mathsf{M} \to \mathsf{N}$ *in the CbV* λ*-calculus iff* $\overline{\overline{\mathsf{M}}}^{\nu} \xrightarrow{*} \overline{\overline{\mathsf{N}}}^{\nu}$.

Proof. By induction on M. If $\mathsf{M} = \mathsf{V}$ is a value (the base of the induction), it is readily seen that $\overline{\overline{\mathsf{V}}}^{n} = *; \underline{\mathsf{V}}_{n}$ and $\overline{\overline{\mathsf{V}}}^{\nu} = *; \underline{\mathsf{V}}_{\nu}$ are in normal form (see the proof of item 2 of Lemma 18). Then, let us prove the inductive steps of the two items in the statement.

1. Let $\mathsf{M} = \mathsf{V}\,\mathsf{M}_1 \dots \mathsf{M}_k$, where V is a value and $k > 0$. By Remark 17, we know that $\overline{\overline{\mathsf{M}}}^{n} = [\overline{\mathsf{M}_1}^{n}, \dots, \overline{\mathsf{M}_k}^{n}]; \underline{\mathsf{V}}_{\nu}$. If $\mathsf{V} = x$ is a variable, M is a CbN normal form and $[\overline{\mathsf{M}_1}^{n}, \dots, \overline{\mathsf{M}_k}^{n}]; x$ is a normal form too. If $\mathsf{V} = \lambda x.\mathsf{P}$ is a λ-abstraction, then $\mathsf{M} \to \mathsf{P}[\mathsf{M}_1/x]\mathsf{M}_2 \dots \mathsf{M}_k = \mathsf{N}$ in the CbN and

$$\begin{aligned}
\overline{\overline{\mathsf{M}}}^{n} &= [\overline{\mathsf{M}_1}^{n}, \dots, \overline{\mathsf{M}_k}^{n}]; \lambda\langle x, k\rangle.k; \overline{\mathsf{P}}^{n} \\
&\to [\overline{\mathsf{M}_2}^{n}, \dots, \overline{\mathsf{M}_k}^{n}]; \overline{\mathsf{P}}^{n}[\overline{\mathsf{M}_1}^{n}/x] \\
&= [\overline{\mathsf{M}_2}^{n}, \dots, \overline{\mathsf{M}_k}^{n}]; \overline{\mathsf{P}[\mathsf{M}_1/x]}^{n} && \text{(by Lemma 20)} \\
&\xrightarrow{*} [\overline{\mathsf{M}_2}^{n}, \dots, \overline{\mathsf{M}_k}^{n}] \circ \overline{\overline{\mathsf{P}[\mathsf{M}_1/x]}}^{n} && \text{(by Lemma 18 and Lemma 4)} \\
&= \overline{\overline{\mathsf{P}[\mathsf{M}_1/x]\mathsf{M}_2 \dots \mathsf{M}_k}}^{n} = \overline{\overline{\mathsf{N}}}^{n}
\end{aligned}$$

2. We have to analyze two cases.

   (a) Let $\mathsf{M} = \mathsf{V}\,\mathsf{V}_1 \dots \mathsf{V}_k$, where $\mathsf{V}, \mathsf{V}_1, \dots, \mathsf{V}_k$ are values and $k > 0$. By Remark 17 we know that $\overline{\overline{\mathsf{M}}}^{n} = [\underline{\mathsf{V}_1}_{\nu}, \dots, \underline{\mathsf{V}_k}_{\nu}]; \underline{\mathsf{V}}_{\nu}$. If $\mathsf{V} = x$ is a variable, M is a CbN normal form and $[\underline{\mathsf{V}_1}_{\nu}, \dots, \underline{\mathsf{V}_k}_{\nu}]; x$ is a normal form too. If $\mathsf{V} = \lambda x.\mathsf{P}$ is a λ-abstraction, in the CbV $\mathsf{M} \to \mathsf{P}[\mathsf{V}_1/x]\mathsf{V}_2 \dots \mathsf{V}_k$ and

$$\begin{aligned}
\overline{\overline{\mathsf{M}}}^{\nu} &= [\underline{\mathsf{V}_1}_{\nu}, \dots, \underline{\mathsf{V}_k}_{\nu}]; \lambda\langle x, k\rangle.k; \overline{\mathsf{P}}^{\nu} \\
&\to [\underline{\mathsf{V}_2}_{\nu}, \dots, \underline{\mathsf{V}_k}_{\nu}]; \overline{\mathsf{P}}^{\nu}[\underline{\mathsf{V}_1}_{\nu}/x] \\
&= [\underline{\mathsf{V}_2}_{\nu}, \dots, \underline{\mathsf{V}_k}_{\nu}]; \overline{\mathsf{P}[\mathsf{V}_1/x]}^{\nu} && \text{(by Lemma 20)} \\
&\xrightarrow{*} [\underline{\mathsf{V}_2}_{\nu}, \dots, \underline{\mathsf{V}_k}_{\nu}] \circ \overline{\overline{\mathsf{P}[\mathsf{V}_1/x]}}^{\nu} && \text{(by Lemma 18 and Lemma 4)} \\
&= \overline{\overline{\mathsf{P}[\mathsf{V}_1/x]\mathsf{V}_2 \dots \mathsf{V}_k}}^{\nu} = \overline{\overline{\mathsf{N}}}^{\nu}
\end{aligned}$$

   (b) Let $\mathsf{M} = \mathsf{P}\mathsf{Q}\mathsf{V}_1 \dots \mathsf{V}_k$, where $\mathsf{V}_1, \dots, \mathsf{V}_k$ are values and Q is not a value. By Remark 17, $\overline{\overline{\mathsf{M}}}^{\nu} = (\lambda x.\overline{\mathsf{P}}^{\nu}[x, \underline{\mathsf{V}_1}_{\nu}, \dots, \underline{\mathsf{V}_k}_{\nu}]) \circ \overline{\overline{\mathsf{Q}}}^{\nu}$. If Q is in normal form for the CbV, then M is in normal form for the CbV. By the induction hypothesis, $\overline{\overline{\mathsf{Q}}}^{\nu}$ is in normal form and (see the proof of Lemma 18) has not the shape $*; p$ for some p; therefore, $\overline{\overline{\mathsf{M}}}^{\nu}$ is in normal form. If $\mathsf{Q} \to \mathsf{Q}'$ in the CbV, then $\mathsf{M} \to \mathsf{P}\mathsf{Q}'\mathsf{V}_1 \dots \mathsf{V}_k = \mathsf{N}$ in the CbV. By the induction hypothesis, $\overline{\overline{\mathsf{Q}}}^{\nu} \to \overline{\overline{\mathsf{Q}'}}^{\nu}$ and $\overline{\overline{\mathsf{M}}}^{\nu} \to$

$(\lambda x.\overline{P}^{v}[x, \underline{V_1}_{v}, \ldots, \underline{V_k}_{v}]) \circ \overline{\overline{Q'}}^{v}$, by Lemma 4. Then, if Q' is not a value, $(\lambda x.\overline{P}^{v}[x, \underline{V_1}_{v}, \ldots, \underline{V_k}_{v}]) \circ \overline{\overline{Q'}}^{v} = \overline{\overline{N}}^{v}$, otherwise, if $Q' = V$ is a value, we have

$$\begin{aligned}
\overline{\overline{M}}^{v} &\to (\lambda x.\overline{P}^{v}[x, \underline{V_1}_{v}, \ldots, \underline{V_k}_{v}]); \underline{V}_{v} \\
&\to \overline{P}^{v}[\underline{V}_{v}, \underline{V_1}_{v}, \ldots, \underline{V_k}_{v}] \\
&\xrightarrow{*} [\underline{V}_{v}, \underline{V_1}_{v}, \ldots, \underline{V_k}_{v}]; \overline{\overline{P}}^{v} \\
&= \overline{\overline{PVV_1 \ldots V_k}}^{v} = \overline{\overline{N}}^{v}
\end{aligned}$$

□

Theorem 22. *For every* λ*-term* M*.*

1. *If* $M \xrightarrow{*} N$ *in the CbN* λ*-calculus, then* $\overline{M}^{n} \xrightarrow{*} \overline{\overline{N}}^{n}$*, namely*

$$\begin{tikzcd}
; \overline{M}^{n} \arrow[r, dotted, ""] & \overline{\overline{N}}^{n} \\
M \arrow[u, mapsto, "\overline{(\cdot)}^{n}"] \arrow[r, "*", "\mathsf{CbN}"'] & N \arrow[u, mapsto, "\overline{\overline{(\cdot)}}^{n}"]
\end{tikzcd}$$

2. *If* $M \xrightarrow{*} N$ *in the CbV* λ*-calculus, then* $\overline{M}^{v} \xrightarrow{*} \overline{\overline{N}}^{v}$*, namely*

$$\begin{tikzcd}
\overline{M}^{v} * \arrow[r, dotted, "*"] & \overline{\overline{N}}^{v} \\
M \arrow[u, mapsto, "\overline{(\cdot)}^{v}"] \arrow[r, "*", "\mathsf{CbV}"'] & N \arrow[u, mapsto, "\overline{\overline{(\cdot)}}^{v}"]
\end{tikzcd}$$

Proof. By Lemma 18 and Proposition 21. □

Theorem 23. *For every* λ*-term* M*.*

1. *If* $*; \overline{M}^{n} \xrightarrow{*} u_{*}$*, then* $M \xrightarrow{*} [\![u_{*}]\!]$ *in the CbN* λ*-calculus, namely*

$$\begin{tikzcd}
; \overline{M}^{n} \arrow[r, ""] \arrow[d, mapsto, "{[\![\cdot]\!]}"] & u_{*} \arrow[d, mapsto, "{[\![\cdot]\!]}"] \\
M \arrow[r, "*", "\mathsf{CbN}"'] & {[\![u_{*}]\!]}
\end{tikzcd}$$

2. If $\overline{\mathsf{M}}^{v}* \xrightarrow{*} \mathsf{u}_*$, then $\mathsf{M} \xrightarrow{*} [\![\mathsf{u}_*]\!]$ in the CbV λ-calculus, namely

$$\begin{tikzcd}
\overline{\mathsf{M}}^{v}* \arrow[r, "*"] \arrow[d, mapsto, "{[\![\cdot]\!]}"] & \mathsf{u}_* \arrow[d, mapsto, "{[\![\cdot]\!]}"] \\
\mathsf{M} \arrow[r, "*", "\mathrm{CbV}"'] & {[\![\mathsf{u}_*]\!]}
\end{tikzcd}$$

Proof. By Lemma 18, $\overline{\mathsf{M}} \xrightarrow{*} \overline{\overline{\mathsf{M}}}$ by a control reduction for CbN and CbV. Therefore, $[\![\mathsf{u}_*]\!] = \mathsf{M}$, for every $\overline{\mathsf{M}} \xrightarrow{*} \mathsf{u}_* \xrightarrow{*} \overline{\overline{\mathsf{M}}}$ (by Proposition 10). By Lemma 18, $\overline{\overline{\mathsf{M}}}$ is either a normal form or a β-redex. When $\overline{\overline{\mathsf{M}}}$ is a β-redex, there is a reduction $\overline{\overline{\mathsf{M}}} \to \mathsf{u}'_* \xrightarrow{*} \mathsf{u}_* \xrightarrow{*} \overline{\overline{\mathsf{N}}}$ s.t. all the rules but the first one are control rules (see the proof of Proposition 21). By Proposition 10, we have then $\mathsf{M} \to [\![\mathsf{u}'_*]\!] = [\![\mathsf{u}_*]\!] = \mathsf{N}$. Proposition 21 ensures that $\mathsf{M} \to [\![\mathsf{u}_*]\!]$ by CbN or CbV according to the case that we are considering. □

6 Conclusions and further work

Starting from the notion of test introduced by Girard in [7], we have proposed a new calculus, the ptq-calculus, in which we reformulate in logical terms the well-known duality programs/continuations, namely in terms of the proofs/tests duality. In the core of the paper we have shown that the ptq-calculus has interesting logical and computational properties and, by encoding λ-calculus Call-by-Value and Call-by-Name into it, we have shown that it might be a fruitful framework for the analysis of reduction strategies and of sequential features of functional programming languages.

In spite of the classical flavour of ptq-calculus, in the paper we have restricted our analysis to the intuitionistic case—mainly beacuse our goal was to present the ptq-calculus as a tool for the study of λ-calculus Call-by-Value and Call-by-Name. The natural extension of the analysis pursued in the paper to classical logic leads to relate our approach to Parigot's $\lambda\mu$-calculus [12, 13]. In particular, there is a natural bijection between ptq-calculus and $\lambda\mu$-calculus that, however, does not give a simulation, namely the reductions of the ptq-calculus are not sound w.r.t. the reductions of the $\lambda\mu$-calculus proposed by Parigot. Such a mismatch reflects the fact that the ptq-calculus is neither Call-by-Value nor Call-by-Name, while with the reduction rules of Parigot the $\lambda\mu$-calculus is essentially Call-by-Name. Therefore, in order to extend our analysis to the classical case, we aim at relating the ptq-calculus with both the original Call-by-Name $\lambda\mu$-calculus proposed by Parigot and to the Call-by-Value $\lambda\mu$-calculus proposed by Ong and Stewart [11], and with Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus [1].

References

[1] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *ICFP '00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 233–243, New York, NY, USA, 2000. ACM Press.

[2] Olivier Danvy. On evaluation contexts, continuations, and the rest of the computation. In Hayo Thielecke, editor, *Proceedings of the Fourth ACM SIGPLAN Continuations Workshop (CW'04)*, number CSR-040-1 in Technical Report. Proceedings of the Fourth ACM SIGPLAN Continuations Workshop (CW'04), Birmingham, UK, 2004.

[3] Philippe de Groote. An environment machine for the lambda-mu-calculus. *Mathematical Structures in Computer Science*, 8(6):637–669, 1998.

[4] Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker, and Bruce F. Duba. Reasoning with continuations. In Albert Meyer, editor, *Proceedings of the First Annual IEEE Symp. on Logic in Computer Science, LICS 1986*, pages 131–141. IEEE Computer Society Press, June 1986.

[5] Michael J. Fischer. Lambda calculus schemata. In *Proceedings of ACM conference on Proving assertions about programs*, pages 104–109, New York, NY, USA, 1972. ACM Press.

[6] Carsten Führmann and Hayo Thielecke. On the call-by-value CPS transform and its semantics. *Inform. and Comput.*, 188(2):241–283, 2004.

[7] Jean-Yves Girard. On the meaning of logical rules i: syntax vs. semantics. In U. Berger and H. Schwichtenberg, editors, *Computational Logic*, volume 165 of *NATO series F*, pages 215–272. Springer, 1999.

[8] Timothy G. Griffin. The formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90, San Francisco, CA, USA, 17–19 Jan 1990*, pages 47–57. ACM Press, New York, 1990.

[9] Martin Hofmann and Thomas Streicher. Continuation models are universal for lambda-mu-calculus. In *LICS '97: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, page 387, Washington, DC, USA, 1997. IEEE Computer Society.

[10] Ichiro Ogata. A proof theoretical account of continuation passing style. In *CSL '02: Proceedings of the 16th International Workshop and 11th Annual Conference of the EACSL on Computer Science Logic*, pages 490–505, London, UK, 2002. Springer-Verlag.

[11] C.-H. Luke Ong and Charles A. Stewart. A Curry-Howard foundation for functional computation with control. In *Conf. Record 24th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'97, Paris, France, 15–17 Jan. 1997*, pages 215–227. ACM Press, New York, 1997.

[12] Michel Parigot. λμ-calculus: an algorithmic interpretation of classical natural deduction. In *Logic programming and automated reasoning (St. Petersburg, 1992)*, volume 624 of *Lecture Notes in Comput. Sci.*, pages 190–201. Springer, Berlin, 1992.

[13] Michel Parigot. Proofs of strong normalisation for second order classical natural deduction. *J. Symbolic Logic*, 62(4):1461–1479, 1997.

[14] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[15] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6:289–360, 1993.

[16] Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Math. Structures Comput. Sci.*, 11(2):207–260, 2001.

[17] Th. Streicher and B. Reus. Classical logic, continuation semantics and abstract machines. *J. Funct. Programming*, 8(6):543–572, 1998.